**Игровые движки, библиотеки и фреймворки, использующие LWJGL и основанные на нем.**

Информация, представленная в этом разделе - это выборочная, переведенная информация. Более подробно, и об остальных движках можете узнать из источника.

**Источники:**

<http://wiki.lwjgl.org/wiki/Game_Engines_and_Libraries_Using_LWJGL.html>

**Несколько фреймворков для 2D разработки:**

**LibGDX | 2D, 3D**

LibGDX – библиотека, использующая OpenGL и позволяющая разрабатывать под 2D и 3D. Поддерживает написание под настольные ПК и Android. Она абстрагирует различия между написанием приложения под настольный ПК и Android на основе OpenGL. Это позволяет разрабатывать приложение полностью на рабочем ПК, а для его запуска на Android потребуется всего 6 строк кода.

**Slick2D | 2D**

Slick2D - это библиотека, предназначенная для создания быстрых, современных аппаратно-ускоренных 2D-игр. Ведется работа по его портированию на Android. Она предоставляет такие инструменты, как: редактор карт, редактор частиц, редактор растровых шрифтов и т.д.

**PlayN | 2D**

PlayN - это кроссплатформенная библиотека для написания игр, которые могут компилироваться в Desktop Java version, в браузерый HTML5, Android, iOS, Flash и другие системы. Библиотека PlayN является бесплатной, с открытым исходным кодом.

**JOGE | 2D**

[Java Opengl Game engine] – движок для создания 2D игр, написанный на Java. Он быстр и очень прост в освоении

**Также перечислю несколько фреймворков для 3D разработки:**

**Ardor | 3D**

Ardor3D - это профессионально ориентированный 3D-движок с открытым исходным кодом на основе Java.

**Clyde | 3D**

Эта библиотека прдоставляет средства для создания сетевых 3D игр на Java.

**JPCT | 3D**

JPCT предлагает средства для разработки игр и приложений для браузеров.

**Ввод с мыши и клавиатуры в LWJGL 2. Input.**

Информация, представленная здесь – выполненный мною перевод документации lwjgl 2 по разделу input.

**Источники:**

<http://wiki.lwjgl.org/wiki/LWJGL_Basics_2_(Input).html>

Также, где-то снабдил информацию дополнительными пояснениями, внес уточнения.

LWJGL может обрабатывать ввод с помощью собственных классов **Keyboard** и **Mouse**.

Для получения обновлений нажатий клавиш или клика требуется обращение (с помощью метода poll()) к классам **Keyboard** и **Mouse**.

В LWJGL это делается автоматически при вызове метода Display.update().

**Мышь.**

Положение курсора мыши на дисплее можно узнать, вызвав методы:



Позиция курсора с координатами (0, 0) соответствует левому, нижнему углу дисплея (в традиционном стиле OpenGL).

Метод **Mouse.isButtonDown(int button)** вернет true, если кнопка мыши нажата, или false, если нет. Кнопки мыши пронумерованы от 0 до buttonCount.



**Клавиатура.**

Метод **Keyboard.isKeyDown(int key)** вернет true, если соответствующая клавиша нажата, или false, если нет.



Ссылку на полный список клавиш клавиатуры найдете ниже, в конце главы.

**Буффер событий.**

Использование методов **isKeyDown()** или **isButtonDown()** может работать хорошо, однако можно пропустить (не зафиксировать) нажатие клавиш или кнопок мыши, если запрос о нажатии (вызов метода **poll()** под капотом) выполняется долго.

Чтобы решить эту проблему, можно использовать буфер событий.

Этот буфер содержит все события от клавиш и кнопок мыши, полученных дисплеем.

Доступ к событиям в буфере осуществляется с помощью методов **getEvent\*()**, где вместо звездочки подставляются слова, характеризующие события, например, **Keyboard.getEventKey().**

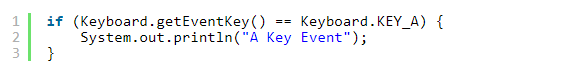
Метод **next()** используется для перебирания буфера событий.

Каждый раз, когда вызывается метод **next()**, он вытаскивает следующее событие из буфера в качестве клавиши/кнопки мыши. После перебора всех событий (клавиш/кнопок мыши) он возвращает false.

Например, буфер клавиатуры можно перебирать так:

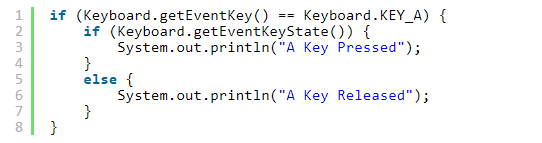


Метод **Keyboard.getEventKey()** возвращает клавишу, которая сгенерировала событие.

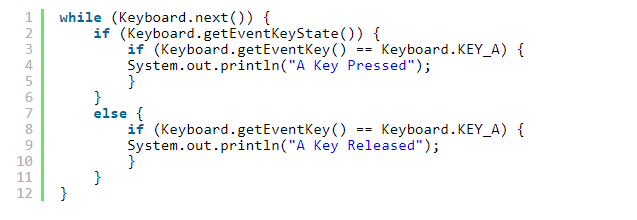


Метод **Keyboard.getEventState()** вернет true, если клавиша события была нажата, или false, если она была отпущена.

Используя два вышеупомянутых метода, можно определить, какая клавиша сгенерировала событие и была ли она нажата или отпущена:



В приведенном ниже примере проверяются события для клавиши A, хранящиеся в буфере.



Так, если запустить этот код в бесконечном цикле рендеринга, то, если мы

1. Зажмем клавишу А, то в консоль выведется «A Key Pressed» (один раз, пока мы ее не отпустим). Далее, спустя 5 сек, если мы отпустим ее, то в консоль выведется «A Key Released».
2. Если клавишу A нажать и сразу же опустить, то почти мгновенно последовательно выведется

«A Key Pressed

A Key Released»

1. Если в результате выполнения кода вывелось только «A Key Pressed», то это означает, что клавиша все еще зажата. Можно в положительную ветку if написать код, который выполнит какую-нибудь логику, если спустя n сек времени «A Key Released» не появился (т.е. клавишу зажали надолго).

**Полный список клавиш клавиатуры можно найти здесь:**

Здесь собраны все константы из всех классов, которые используются в LWJGL (в том числе и константы для клавиатуры, с их числовыми значениями):

<http://legacy.lwjgl.org/javadoc/constant-values.html>

Или погугля, введя в поиск что-то вроде «javadoc list of Keyboard keys lwjgl», если ссылки не будут работать. Список там огромный, на пару десятков страниц, поэтому не стал его сюда добавлять.

**Документация по классу Keyboard** (здесь вы также найдете константы-клавиши, а также дополнительные методы этого класса):

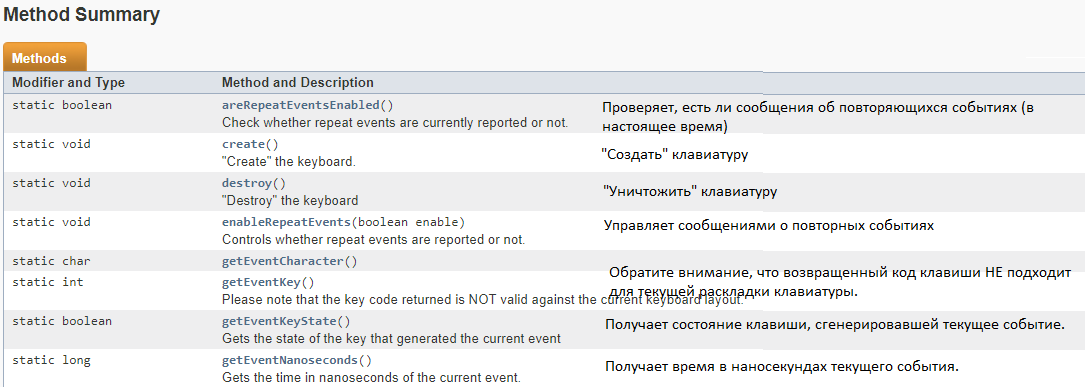
<http://legacy.lwjgl.org/javadoc/org/lwjgl/input/Keyboard.html>

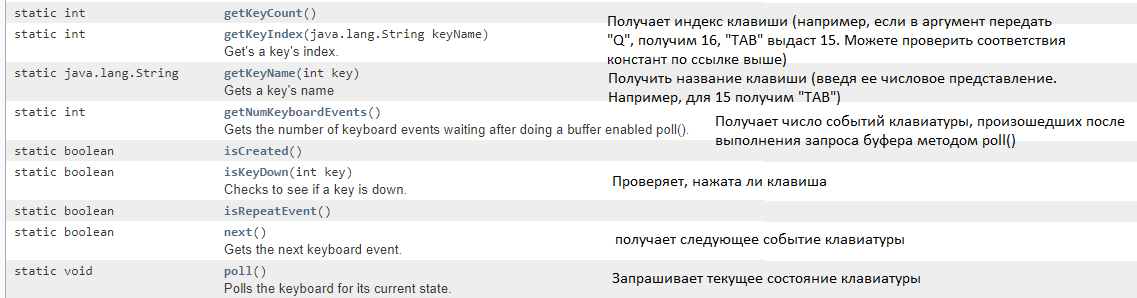
**Документация по классу Mouse** (здесь вы найдете константы, а также дополнительные методы этого класса):

<http://legacy.lwjgl.org/javadoc/org/lwjgl/input/Mouse.html>

Методы класса Keyboard и Mouse, пожалуй, переведу (чтобы были под рукой).

**Keyboard:**





**Mouse:**

